DB Name	Query	Hit Count	Set Name
USPT	(global adj register$) and (local adj register$) and (plurality with processors)	15	L13
USPT	l7 and (global adj register$) and (local adj register$) and (plurality with processors)	0	L12
USPT	l8 and vliw	0	L11
USPT	l9 and vliw	0	L10
USPT	l8 and (plurality with processors)	1	L9
USPT	l7 and ((global and local and register$).clm.)	3	L8
USPT	eng$.xp.	1001	L7
USPT	5657291.pn.	1	L6
USPT	5721868.pn.	1	L5
USPT	5761475.pn.	1	L4
USPT	5764943.pn.	1	L3
USPT	5778248.pn.	1	L2
USPT	4155119.pn.	1	L1

L13 #5

WEST

Generate Collection

L13: Entry 5 of 15 File: USPT Jan 7, 1997

DOCUMENT-IDENTIFIER: US 5592679 A
TITLE: Apparatus and method for distributed control in a processor architecture

ABPL:
The present invention provides a multi-level instruction scheduling system for controlling multiple execution pipes of a distributed data flow (DDF) processor. The multi-level scheduling system includes a simple global instruction scheduler and multiple local instruction schedulers corresponding to the number of execution pipes. The global instruction scheduler is responsible for distributing instructions among the execution pipes. Each local instruction scheduler is only responsible for scheduling its share of distributed instructions and matching the reduced number of instructions with the execution units of the corresponding execution pipe when all its source operands are available. Source operands are garnered in one of three ways. First, the local instruction scheduler ensures that locally generated register operand values are stored in a local register buffer and made available quickly to younger instructions distributed to the execution pipe. Second, when source operand values of the instruction are not available in the local register buffer, an inter-pipe operand request is made to an arbiter. If the operand value(s) is available from another execution pipe, a transfer of the source operand(s) is initiated via an inter-pipe bypass coupling the first and second execution pipes. Third, if the source register operand value(s) cannot be found in any of the other execution pipes, the register operand value(s) is retrieved from a global register file via an inter-pipe bypass. This multiple execution pipe architecture advantageously lends itself to software optimizing techniques such as multi-tasking, system exception/trap handling and speculative execution, e.g., instruction branch prediction techniques.

BSPR:
FIG. 1 is a block diagram showing a conventional data flow processor architecture having multiple execution units. Data flow processor 100 includes an instruction cache memory 110, a prefetch buffer 120, a centralized instruction scheduler 130 with a speculative register file 135, a plurality of execution units 141, 142, . . . 149, and a register file/memory 190. Prefetched buffer 120 is coupled between instruction cache memory 110 and centralized instruction scheduler 130. Register file/memory 190 is coupled to centralized instruction scheduler 130. Each of the plurality of execution units 141, 142, . . . 149 are coupled to centralized instruction scheduler 130.

BSPR:
First, the local instruction scheduler ensures that locally generated register operand values are stored in a local register buffer to be used by younger instructions distributed to the execution pipe. Hence, locally generated register result values are made quickly available locally, i.e., within each execution pipe, without the need for external access. In other words, intra-pipe operand interdependencies are resolved locally and quickly by the local instruction scheduler.

BSPR:
Second, when source operand values of the instruction are not available in the local register buffer, an inter-pipe operand request is made to an arbiter. The arbiter which is responsible for resolving inter-pipe operand interdependencies, may broadcast the request to all or a subset of the execution pipes. Alternatively, the arbiter may interrogate the global instruction scheduler to extract the identity of another execution pipe with the required source operand(s). In other embodiments, the source(s) of the needed operand(s) of each instructions, i.e., the identity of the execution pipe(s), is dispatched together with the instruction, thereby simplify the subsequent garnering of the source operand value(s). When the operand value(s) is available from another execution

pipe, a transfer of the source operand(s) is initiated via an inter-pipe bypass coupling the first and second execution pipes.

BSPR:
Third, if the source register operand value(s) cannot be found in any of the other execution pipes, the register operand value(s) is retrieved from a global register file. The requested operand value is then transferred via the inter-pipe bypass coupling the first execution unit to the global register file.

BSPR:
When all the source operand values have been transferred to the execution pipe, the instruction is matched with an available execution unit in the execution pipe. Eventually, the instruction completes execution in the execution pipe and is retired. The result operand value generated by the execution pipe is written into the global register file. In addition, the local instruction scheduler communicates the completion of execution to the global instruction scheduler, thereby completing the processing of the instruction.

DEPR:
FIG. 2 is a block diagram illustrating the DDF processor architecture in accordance with one embodiment of the present invention. DDF processor 200 includes an instruction cache memory 210, a plurality of prefetch buffers 221, 222, . . . 229, a global instruction scheduler 230, a plurality of execution pipes 241, 242, . . . 249, one or more inter-pipe bypass(es) 251, . . . 259, an arbiter 260, and a global register file/memory 290. Each instruction pipe includes its own local instruction buffer, local instruction scheduler, execution unit(s) and temporary local register buffer. For example, instruction pipe 241 includes a local instruction buffer 241a, a local instruction scheduler 241b, execution unit(s) 241c and a local register buffer 241d. Similarly, instruction pipe 242 includes a local instruction buffer 242a, a local instruction scheduler 242b, execution unit(s) 242c and a local register buffer 242d. Hence a description of the operation of execution pipe 241 is equally applicable the other execution pipes 242, . . . 249.

DEPR:
In accordance with one aspect of the invention, DDF processor 200 provides a hierarchical multi-level scheduling scheme for processing a plurality of instructions. Global instruction scheduler 230 is responsible for the top level scheduling which involves allocating and distributing instructions among execution pipes 241, 242, . . . 249. In contrast, local instruction schedulers 241b, 242b, . . . 249b, are responsible for the lower level scheduling which involves matching instructions with execution units and garnering source register operand values for instructions in execution pipes 241, 242, . . . 249, respectively.

DEPR:
In accordance with another aspect of the present invention, in order to facilitate rapid access to register operand values, local instruction scheduler 241b stores locally generated register operand values in local register buffer 241d for use as source operands values by younger instructions distributed to execution pipe 241. As a result, all locally generated register register result values are made available locally, i.e., within execution pipe 241, without the need to access outside of execution pipe 241. In other words, intra-pipe operand interdependencies are resolved locally by local instruction scheduler 241b. During normal program execution, invalidation of operand values stored in register buffer 241d by local instruction scheduler 241b is not required because the respective registers in register buffer 241d are renamed prior to reuse.

DEPR:
The hierarchical register file system which includes global register file 290 and multiple local register buffers 241d, 242d, . . . 249d is advantageous over the prior art for the following reasons. A single centralized register file is not ideal because interconnecting the large fast global memory to a large number of execution units is prohibitively expansive in silicon area and requires extremely complex circuitry with many I/O ports. This expansion increases exponentially as the size of the global memory and/or the number of execution units increase.

DEPR:
Conversely, by dedicating a small number of fast register buffers located close

to a small subset of execution units as in the present invention, faster access times to the most frequently accessed operands are possible without incurring a high penalty in silicon area. This is because statistically the majority (e.g., >90%) of source operands are generated by other instructions which are spaced less than ten instruction lines away in program order. Hence, instead of a single global register file, local register buffers 241d, 242d . . . 249d efficiently handle the bulk of the operand accesses without prohibitively increasing the overall circuit complexity of DDF processor 200. As a result, the hierarchical register file system is easily scalable. Note that the above described hierarchical register file organization is equally applicable to other processor architectures having multiple execution units, e.g., a single pipeline processor with multiple execution units.

DEPR:
In the (less likely) event that one or more source operand(s) of an instruction in execution pipe 241 cannot be found in local register buffer 241d, an inter-pipe operand request is made by execution pipe 241 to arbiter 260. Arbiter 260 which is responsible for resolving inter-pipe operand interdependencies, initiates a request for the identity of the execution pipe with the required source operand(s). If the operand value(s) is located in another execution pipe, upon receiving the identity of the execution pipe with the source operand, e.g., pipe 242, from global instruction scheduler 230, arbiter 260 initiates a transfer of the source operand(s) via the appropriate one of inter-pipe bypass(es) 251, . . . 259. Note that generally, inter-pipe bypasses are required for interconnecting execution pipes with similar classes of execution unit(s). For example, a first and a second pair of execution pipes 241, 242, . . . 249 can be floating point execution pipes and integer execution pipes, respectively, with a first bypass interconnecting the pair of floating point execution pipes and a second bypass interconnecting the pair of integer execution pipes.

DEPR:
In the (even less likely) event that the source operand value(s) cannot be located in any of execution pipes 241, 242, . . . 249, then a request to access global register file 290 is initiated. The requested register operand value is transferred from global register file 290 into execution pipe 241. If the instruction has all needed source operand values, it is now ready to matched to an execution unit of execution pipe 241.

DEPR:
Once all the source operands of each instruction in execution pipe 241 has been retrieved either from within the same pipe 241, from one of the other execution pipes 242, . . . 249 via inter-pipe bypass 251, and/or from global register file 290, local instruction scheduler 241b matches and dispatches the instruction to a free execution unit of execution unit(s) 241c. Local register buffer 241d of execution pipe 241 advantageously allows faster local access to register values produced within execution pipe 241. Eventually, the instruction completes execution in execution pipe 241 and is retired. The result operand value generated by execution pipe 241 is then written into global register file 290. In addition, local instruction scheduler 241b communicates the completion of execution to global instruction scheduler 230, thereby completing the processing of the instruction.

DEPR:
The multiple prefetch buffer and multiple execution pipe architecture of DDF processor 200 also lends itself to software and system optimizing techniques such as multi-tasking, system exception/trap handling and speculative execution, e.g., instruction branch prediction techniques. For example, DDF processor 200 can perform multi-tasking by segregating each task in a separate one of prefetch buffers 221, 222, . . . 229 and concurrently executing individual tasks in a separate one of execution pipes 241, 242, . . . 249. Although multiple sets state may be needed to support multi-tasking, hardware such as local register buffers 241d, 242d, . . . 249d dedicated to execution units 241c, 242c, . . . 249c, respectively, and inter-pipe bypasses 251, . . . 259 for sharing data and communicating between multiple execution tasks/paths already exist.

CLPR:
7. The method of claim 4 wherein said providing step includes the step of retrieving said first source operand value from a local register buffer of said first execution pipe.

CLPR:
8. The method of claim 4 wherein said garnering step includes the step of retrieving said second source operand value from a local register buffer of said second execution pipe.

CLPV:
an inter-pipe bypass coupled to said first and said second execution pipe for transferring said local register operand values between said first and said second execution pipe;

CLPV:
an arbiter coupled to said inter-pipe bypass for garnering one of said local register operand values from said first execution pipe to said second execution pipe when said one local register operand value of said first execution pipe is useful as a source operand value for said second execution pipe; and

CLPV:
a global register file coupled to said arbiter for storing global register operand values, said arbiter garnering one of said global register operand values from said global register file to said first execution pipe when said one global register operand value is useful as a source operand value for said first execution pipe, wherein said global register file is coupled to said inter-pipe bypass and said one global register operand value is garnered from said global register file to said first execution pipe via said inter-pipe bypass.

CLPV:
providing a local register operand value generated by said first execution pipe for use as a first source operand value by said first execution pipe;

CLPV:
garnering a local register operand value generated by said second execution pipe to said first execution pipe when said local register operand value of said second execution pipe is useful as a second source operand value for said first execution pipe, said local register operand value garnered via an inter-pipe bypass coupled to said first and second execution pipes; and

CLPV:
garnering a global register operand value from a global register file to said first execution pipe when said global register operand value is useful as a third source operand value for said first execution pipe, said global register file is coupled to said inter-pipe bypass and said global register operand value is garnered from said global register file to said first execution pipe via said inter-pipe bypass.

CLPW:
a local register buffer coupled to said at least one execution unit for storing local register operand values generated by said at least one execution unit; and

CLPW:
a local instruction scheduler for scheduling said corresponding one sub-group of instructions, said local instruction scheduler providing said local register operand values to said at least one execution unit when said local register operand values are useful as source operand values by said at least one execution unit;

L13: Entry 6 of 15 File: USPT Feb 7, 1995

DOCUMENT-IDENTIFIER: US 5388235 A
TITLE: Arithmetic and logic processor and operating method therefor

BSPR:
FIG. 3A shows a constitution example of a register window which can be accessed by one procedure. The register window includes global registers at logic register numbers R0-R7, low registers at logic register numbers R8-R15, local registers at logic register numbers R16-R23, and high registers at logic register numbers R24-R31. The global registers store global variables used commonly in all procedures. The low registers and the high registers are regions where parameters are transmitted and received between procedures. The local registers are registers which an associated procedure uses dedicatedly. Now, operation will be described in the case that procedure A is executed and calls procedure B, and subsequently the procedure B calls procedure C as shown in FIG. 3B.

BSPR:
FIG. 4A shows specific constitution of a register file. As shown in FIG. 4A, the register file is constituted by global register regions of physical register numbers 0-7 and local register regions of physical register numbers 8-119. The local register regions of physical register numbers 8-119 are divided into unit regions (window) each comprising low register region, local register region and high register region, and a divided region unit is assigned to a procedure. Now assume that procedure A shown in FIG. 3B is under execution and uses the register file in the regions of the physical addresses (physical register number) 0-7 as global registers and the physical addresses 8-31 as local registers. When the procedure A calls procedure B, the procedure A stores arguments (parameter p1, . . . p6) to the register file (high a) of the addresses 24-31 and then calls the procedure B. The called procedure B uses the register file in the addresses 0-7 as global registers and the addresses 24-47 as local registers. Subsequently when the procedure B calls procedure C, the procedure B stores arguments (q1, . . . Q6) to the registers of the numbers (address) 40-47 and then calls the procedure C. The called procedure C uses the register file in the addresses 0-7 as global registers and the addresses 40-63 as local registers. That is, explaining the procedure B as an example, as shown in FIG. 4B, the high registers R24-R31 used by the procedure B are physically the same as the local registers R8-R15 used by the procedure C. Also the low registers R8-R15 used by the procedure B are physically the same as the high registers R24-R31 used by the procedure A. The transfer of parameters between the procedures can be performed on the registers being made physically the same. Consequently, the parameters need not be transferred to the memory, and the procedure can be called at high speed.

CLPR:
1. An arithmetic and logic processor having a register file and a first memory, wherein said register file includes a plurality of register windows one assigned to each procedure, and each of the plurality of register windows includes a plurality of registers, said processor comprising:

CLPR:
4. An arithmetic and logic processor having a register file and a first memory, wherein said register file includes a plurality of register windows one assigned to each procedure, and each of the plurality of register windows including a plurality of registers, said processor comprising:

CLPR:
6. An arithmetic and logic processor having a register file with a plurality of register windows, a stack memory and a first memory, each of said plurality of register windows including registers corresponding to a prescribed number of words and one register window being assigned to one procedure, said processor comprising:

CLPR:
8. Data processing method in an arithmetic and logic processor having a register file with a plurality of register windows and a first memory, wherein registers corresponding to a predetermined number of words are assigned to each register window, said method comprising the steps of:

CLPR:
11. Data processing method in an arithmetic and logic processor having a register file with a plurality of register windows and a first memory, wherein registers corresponding to a predetermined number of words are assigned to each register window, said method comprising the steps of:

WEST

Generate Collection

L13: Entry 2 of 15 File: USPT Feb 8, 2000

DOCUMENT-IDENTIFIER: US 6023757 A
TITLE: Data processor

BSPR:
The VLIW type processor is a processor forming an instruction system with an instruction format of long instruction length though fixed length. In such a VLIW processor, a plurality of computing units are simultaneously operated by means of an instruction of 128 bits or longer. The number of computing units (the number of processing instructions executable in parallel) to be simultaneously operated is greater than that of the superscalar system. In the VLIW type processor, instruction dependency is checked beforehand at the time of compiling the source program and simultaneously executable instructions are combined into one instruction. When the number of simultaneously executable instructions does not reach the maximum number of simultaneously executable instructions, a NOP (No-operation) instruction is inserted by a compiler. The circuit scale of the VLIW type processor is smaller than that of the superscalar system of the RISC type microprocessor because the instruction dependency need not be checked at the time of execution of the VLIW type processor. Trace ce/300 of multiflow Computer Co. carried in Nikkei Electronics of Nov. 27, 1989 (No. 487), pp 196-197 is a typical example of the VLIW type processor.

BSPR:
The data processor includes a first data processing unit which has a plurality of computing units and is used for reading an instruction from a first memory which stores a first group of instructions and executing the instruction, a second data processing unit which has a plurality of computing units capable of parallel operation and is used for executing instructions simultaneously executable as one instruction, and a second memory for storing a group of instructions which the second data processing unit executes. When the instruction read by the first data processing unit is a predetermined instruction, an instruction to be executed by the second data processing unit is read from the second memory.

DEPR:
The address of the instruction address bus 105 is used to read the VLIW instruction from the VLIW table 113 to a VLIW instruction bus 112. The VLIW instruction decode unit 115 decodes the VLIW instruction on the VLIW instruction bus 112 on the basis of the activating condition transferred from the VLIW start decision unit 104 through a VLIW instruction decode start signal 111, and individually controls the arithmetic units 120 or the integer arithmetic unit 119 and floating-point arithmetic unit 118 of the RISC core section 106 in parallel through a VLIW arithmetic unit control signal 116. The arithmetic units 120 include an integer arithmetic unit, a floating-point arithmetic unit, a DSP (Digital Signal Processor: e.g., product sum computation) dealing with fixed-point data and the like. FIG. 1 shows an example of a local register file type processor when arithmetic units each have their own register files. A transfer of data between arithmetic units is conducted through a computing unit-to-computing unit data bus 123. When the memory access instruction is contained in the VLIW instruction, the VLIW load/store unit 131 has access to the data cache 129 via an address bus 125, a data bus 130 and a data bus 124 for VLIW.

DEPR:
FIG. 2 shows a modified example of the microprocessor of FIG. 1. What is shown in FIG. 2 is a global register type microprocessor having the registers of the RISC and VLIW cores 106, 140 in common, wherein like reference characters designate like component parts of the microprocessor of FIG. 1. The technology of manufacturing semiconductor integrated circuits is used to form a microprocessor 200 on one semiconductor substrate of single crystal silicon and the resin molding technique is used for the packaging of the semiconductor substrate by

so-called plastic packaging.

DEPR:
The processor 1 (708) includes the instruction fetch unit 101, the instruction cache 107, the RISC instruction decode unit 114, one or a plurality of arithmetic units 704 and the RISC load/store unit 127, these being mutually connected by an internal bus 702. The instruction fetch unit 101 reads an instruction from the instruction cache 107 and the RISC instruction decode unit 114 analyzes the instruction thus read, causing the arithmetic units 704 to operate, whereby the RISC load/store unit 127 is activated whenever memory access is needed. A multiplication unit, an integer arithmetic unit, a floating-point unit, a substraction unit, a DSP and so forth are placed in the arithmetic unit 704. The arithmetic unit may have a plurality of similar arithmetic units. A bus 709 is used to input an instruction to the instruction cache from a bus control unit 712. The RISC load/store unit 127 uses the bus 705 to access the data cache 129 and the bus control unit 712 uses the bus 707 to fetch data into the data cache 129.

DEPR:
The processor 2 (711) includes the VLIW start decision circuit 104, the VLIW table 113, the VLIW instruction decode unit 115, one or a plurality of arithmetic units 704 and the VLIW load/store unit 131, these being mutually connected by the internal bus 702. When the activation of the VLIW instruction is decided by the VLIW start decision circuit 104, it is read from the VLIW table 113 and analyzed in the VLIW instruction decode unit 115 and the arithmetic units 704 operate, whereby the VLIW load/store unit 131 is activates whenever memory access is needed. For the VLIW table 113, use can be made of a non-volatile memory such as a masked ROM and a flash memory or a volatile memory such as a SRAM and a DRAM. When the VLIW table is located outside the chip, the VLIW instruction is read from outside via a bus control unit. A multiplication unit, an integer arithmetic unit, a floating-point arithmetic unit, a substraction unit, a DSP and so forth are placed in the arithmetic unit 704. The arithmetic unit may have a plurality of similar arithmetic units. The VLIW load/store unit 131 uses a bus 706 to gain access to the data cache 129.

CLPW:
wherein the execution of the instruction of the second group of instructions in the processor is prohibited and a plurality of the instructions of the first set of instructions equivalent to the instruction of the second group of instructions are executed if said register stores information prohibiting the execution of the instruction of the second group of instructions in the processor.

CLPW:
wherein the execution of the instruction of the second group of instructions in the processor is prohibited and a plurality of the instructions of the first set of instructions equivalent to the instruction of the second group of instructions are executed if said register stores information prohibiting the execution of the instruction of the second group of instructions in the processor.

L13: Entry 8 of 15 File: USPT Feb 16, 1993

DOCUMENT-IDENTIFIER: US 5187791 A
TITLE: Microprocessor with improved interrupt response with data saving dependent upon processor status using status flag

DEPR:
Each of the four identical and independent PUs 12a-d of CPU 10 is a 32-bit RISC (Reduced Instruction Set Computer). The four PUs access the instruction and data caches via interconnection networks 22 and 24. In addition to providing PU cache data transfer paths, these networks provide a direct inter-PU communication path for broadcast operations and global register access, as well as a path for interrupt routing. Instruction and data caches are divided into four banks, and each interconnection network includes a 5.times.4 crossbar switch, permitting simultaneous instruction and data accesses by all four PUs.

DEPR:
An address space is defined by a set of virtual-to-real page mappings which are recorded in a translation table. Each address space has its own translation table. At any instant, only one address space can be active on a CPU; the four PUs always execute in the same address space. A global register holds a pointer to the start of the translation table for the currently active address space. Translation tables have a simple, two-level structure, composed of a first-level directory and one or more second-level page tables. In addition to virtual-real mappings, translation table entries identify pages as system, read-only, non-cacheable, or interrupt-on-write.

DEPR:
The PUs have a small register-oriented instruction set in which all data access to memory is done by register load and store instructions. Register and word size is 32 bits. Each PU 12a-d has 16 general-purpose registers, a total of 64 for CPU 10, and 7 local registers. Local registers include product, remainder, prefix, and various state saving registers. In addition, the four PUs share 8 global registers, including interrupt, event counter, and global status registers.

DEPR:
The 16-bit instruction length limits the size of immediate and displacement fields in the instructions. However, a large proportion of immediate and displacement values encountered in programs are small enough to be contained in these fields. When necessary, larger values can be created by prefixing the immediate or displacement field value. Each PU has a local register called the Prefix Register, whose state (empty or not empty) is represented by a Prefix Valid flag. Values are loaded into the Prefix Register by a Prefix instruction. If the Prefix Register is empty when a Prefix instruction is executed, the immediate field of the Prefix instruction is stored in the low-order bits of the Prefix Register and sign extended, and the Prefix Valid flag is set to not empty. If a second Prefix instruction is then executed, the contents of the Prefix Register are shifted left and the immediate field of the second Prefix instruction is stored in the low-order bits of the Prefix Register. When an instruction with a prefixable immediate or displacement is executed, the Prefix Valid flag is examined. If the Prefix Register is not empty, the contents of the Prefix Register are concatenated with the instruction's immediate or displacement field to form the effective immediate or displacement value. Prefixing also is used to define fields for field manipulation instructions.

DEPR:
The programming model comprises a general register set, status register and program counters, a special register set, and the instruction and data caches. Each PU has its own general register set, status register, and program counters; these registers are said to be local to the PU. Each PU also has its own copy of certain special registers, while other special registers are common to all PUs;

Array from left to right would be Word3 - bit31, Word1 - bit31, Word0 - bit31, Word2 - bit31, and then the next bit, Word3 - bit30, etc. This can be visualized as all four registers stacked one on top of the other, with LSB's on the same end.

DETL:
TABLE I _____ Register Set Register Address
_____ 16 Frame Registers R0-R15 000XXXX 16
Global Registers G0-G15 001XXXX 32 Literals 0 . . . 31 01XXXXX 32 Special Function SF0-SF31 10XXXXX 32 Scratch Registers S0-S31 11XXXXX

_____

CLPR:
3. A data processor connected to a main memory and a micorinstruction bus for carrying a current microinstruction, said current micro instruction including source operands, said data processor including a register array comprised of a plurality of registers; said array including busy bits associated with each register in said array, said data processor including single cycle coprocessors and multi-cycle processors, the method comprising the steps of:

CLPV:
a Scbok line (102) connected to said register array and to first ones of said single cycle coprocessors and multi-cycle processors, said scbok line when driven to a first state, indicating that a first particular register of said plurality of registers or a single cycle coprocessor or a multi-cycle processor used by a current register type of microinstruction on said microinstruction bus is available for a register execution operation and, when driven to a second state, indicating that said first particular register or said single cycle coprocessor or said multi-cycle processor used by said current register type of microinstruction on said microinstruction bus is not available for a register execution operation:

WEST

Generate Collection

L13: Entry 9 of 15 File: USPT Feb 9, 1993

DOCUMENT-IDENTIFIER: US 5185872 A
TITLE: System for executing different cycle instructions by selectively bypassing scoreboard register and canceling the execution of conditionally issued instruction if needed resources are busy

DEPR:
The register file (RF) has 16 local and 16 global registers. It has a small number of scratch registers used only by microcode It also creates the 32 literals (0-31 constants) specified by the architecture. The RF has 4 independent read ports and 2 independent write ports to support the machine parallelism. It also checks and maintains the register scoreboarding logic.

DEPR:
The Memory Interface Unit (9) includes an Address Generation Unit (AGU) and a Local Register Cache (LRC). The AGU does the effective address calculations in parallel with the integer execution unit. It performs the load-effective-address instructions (LDA) and also does the address computations for loads and stores. It has a 32-bit carry-look-ahead adder and a shifter in front of the adder to do the prescaling for the scaled index addressing modes.

DEPR:
The Local Register Cache (LRC) maintains a stack of multiple 16-word local register sets. On each call the 16 local registers are transferred from the register file (RF) to the LRC. This allocates the 16 local registers in the RF for the called procedure. On a return the 16 words are transferred back into the RF to the calling procedure. The LRC uses a single ported RAM cell that is much smaller than the 6-ported RF cell. This keeps the RF small and fast so it can operate at a high frequency while allowing 8+ sets of local registers to be cached on-chip. With this LRC the call and return instructions take 4 clocks.

DEPR:
At any point in time, thirty-two 32-bit registers and four 80-bit floating-point registers are addressable (the 32 registers can also be used to hold floating-point values). Of the 32 registers, 16 are global registers (21) and 16 are local registers. The difference is that the 16 global registers are unaffected when crossing procedure boundaries (i.e., they behave like "normal" registers in other architectures), but the local registers are affected by the call and return instructions.

DEPR:
When a call instruction is executed, the processor allocates to the called procedure a new set of 16 local registers from an on-chip pool of register sets. If the processor's fourset pool is depleted, the processor automatically reallocates a register set by taking one register set associated with an earlier procedure and saving the contents of that register set in memory. The contents of the earlier procedure's register set are saved in the first 16 words of that procedure's stack frame in memory. The return instruction causes the current local register set to be freed (for use by a subsequent call). This mechanism is called the stack frame cache (23) and is more fully described in U.S. Pat. No. 4,811,208 of Meyers et al., entitled "Stack Frame Cache on a Microprocessor Chip" granted on Mar. 7, 1989, and assigned to Intel Corporation, the assignee of the present invention.

DEPR:
The Ram Array entails a 9 row .times.4 32-bit word register ram, which houses the 36 registers used by the processor. These 36 registers include 16 global registers, 16 frame (local) registers, and 4 microcode scratch registers. As mentioned previously, the four registers per row are arranged with like word bits grouped together. For example, at the far left the bits coming out of the Ram

these are called global registers.

DEPR:
The intended interpretation of the PUA flag is that, when set, the kernel is not required to save PU state on interrupt/trap recognition or restore state prior to returning from processing the interrupt, thereby reducing interrupt processing overhead. For example, when switching address spaces, local register saving and restoring, except for the SaveR and PCQ registers as described above, is assumed to be unnecessary if PUA is set. To effect this, PU activities should, on completion of execution, set PUA prior to halting. From a hardware standpoint, the PUA flag is used in the kernel entry address selection and may also be used in selecting a PU to process an external or event counter overflow interrupt.

CLPR:
2. The method of claim 1 wherein said data processing device includes a plurality of programmable processors each having a corresponding processor status flag and wherein execution of said interrupt processing program comprises selection of one of said plurality of processors to execute said interrupt processing program, said selected processor being one with its respective processor status flag set to the first logical state if any of said plurality of processors are idle.

WEST

Generate Collection

Search Results - Record(s) 1 through 1 of 1 returned.

☐ 1. Document ID: US 6023757 A

L14: Entry 1 of 1 File: USPT Feb 8, 2000

DOCUMENT-IDENTIFIER: US 6023757 A
TITLE: Data processor

ABPL:
A data processor which includes a first processor for executing a first instruction set and a second processor for executing a second instruction set different from the first instruction set. When the first processor executes a predetermined instruction of the first instruction set the second processor executes an instruction of the second instructions set. The first processor may be a reduced instruction set computer (RISC) type processor, the second processor may be a very long instruction word (VLIW) type processor, the first instruction set may be a RISC instruction set and the second instruction set may be a VLIW instruction set. The predetermined instruction of the RISC instruction set executed by the first processor may be a branch instruction causing a branch to a specific address space at which VLIW instructions are stored. Thereafter, the VLIW instructions at the specific address space are executed by the VLIW type processor.

BSPR:
There are varieties of architectures for microprocessors. For example, there are microprocessors of CISC (Complexed Instruction Set Computer), RISC (Reduced Instruction Set Computer) and VLIW (Very Long Instruction Word) (or LIW (Long Instruction Word)) types to name a few.

BSPR:
The VLIW type processor is a processor forming an instruction system with an instruction format of long instruction length though fixed length. In such a VLIW processor, a plurality of computing units are simultaneously operated by means of an instruction of 128 bits or longer. The number of computing units (the number of processing instructions executable in parallel) to be simultaneously operated is greater than that of the superscalar system. In the VLIW type processor, instruction dependency is checked beforehand at the time of compiling the source program and simultaneously executable instructions are combined into one instruction. When the number of simultaneously executable instructions does not reach the maximum number of simultaneously executable instructions, a NOP (No-operation) instruction is inserted by a compiler. The circuit scale of the VLIW type processor is smaller than that of the superscalar system of the RISC type microprocessor because the instruction dependency need not be checked at the time of execution of the VLIW type processor. Trace ce/300 of multiflow Computer Co. carried in Nikkei Electronics of Nov. 27, 1989 (No. 487), pp 196-197 is a typical example of the VLIW type processor.

BSPR:
The adoption of the RISC type architecture, for example, has improved operating frequency and increased the number of simultaneously executable instructions, thus resulting in improved performance of microprocessors. It has therefore been planned to use software for compression/expansion (MPEG (Moving Picture Experts Group(decoder/encoder) of moving pictures and three-dimensional graphic processing that have heretofore been done by using dedicated hardware or controllers.

BSPR:

The VLIW-type microprocessor is thought to be fit for a processing for multimedia use dealing with a large quantity of data by repeating the same process including MPEG decoder/encoder and three-dimensional graphic processing.

BSPR:
A typical conventional microprocessor or a CPU (Central Processing Unit) is of a CISC or RISC type and possesses an accumulation of software assets. Microprocessors of the CISC, RISC and VLIW types each have different instruction sets, instruction formats, addressing modes, programming modes and the like; that is, no software compatibility exits.

BSPR:
The data processor includes a first data processing unit which has a plurality of computing units and is used for reading an instruction from a first memory which stores a first group of instructions and executing the instruction, a second data processing unit which has a plurality of computing units capable of parallel operation and is used for executing instructions simultaneously executable as one instruction, and a second memory for storing a group of instructions which the second data processing unit executes. When the instruction read by the first data processing unit is a predetermined instruction, an instruction to be executed by the second data processing unit is read from the second memory.

BSPR:
The present invention further provides a microprocessor which includes a RISC core for executing RISC instruction, a VLIW table for storing VLIW instructions, a VLIW core for executing a VLIW instruction, and a VLIW start decision unit for controlling operation switching between the RISC core and the VLIW core. When the RISC core executes a subroutine call or a branch instruction out of RISC instructions, a VLIW instruction is read from the VLIW table by use of a branch destination address and operation is transferred by the VLIW start decision unit from the RISC core to the VLIW core, whereby the VLIW instruction is caused to operate until a VLIW core completion code is read from the VLIW table.

BSPR:
The microprocessor further includes an address translation table translation-lookaside-buffer (TLB) for subjecting the branch destination address to address translation so that a signal for expanding the VLIW instruction read from the VLIW table may be read from the TLB simultaneously when the address translation is conducted.

BSPR:
The microprocessor even further includes a circuit for examining whether the operation of the VLIW instruction executed subsequent to a branch to the VLIW instruction is relevant to the operation of the RISC instruction which is restored after the execution of the VLIW instruction. The microprocessor is capable of executing the RISC instruction at the restoring destination in parallel without waiting for the completion of the VLIW instruction when no relevance exists.

BSPR:
The microprocessor includes a register for prohibiting and controlling the execution of the VLIW instruction in the processor and a function of lowering power consumption by causing a branch to a string of RISC instructions performing an operation equivalent to the VLIW instruction at the time the execution of the VLIW instruction is prohibited so as to suspend the operation of the VLIW core.

BSPR:
A non-volatile memory has a built-in VLIW table. The non-volatile memory is preferably one of a masked ROM, a flash memory and a ferroelectric memory. A volatile memory has a built-in VLIW table. The volatile memory of the VLIW table is preferably either SRAM or DRAM. A memory rewritable with the processor of the RISC core preferably has a built-in VLIW table.

DRPR:
FIG. 3 is a detailed illustration of the VLIW start theory of the microprocessor in FIGS. 1 and 2;

DRPR:
FIG. 4 is a detailed illustration of the VLIW start theory when a translation

table in a translation-lookaside-buffer (TLB) is used for the microprocessor in FIGS. 1 and 2;

DRPR:
FIGS. 5A and 5B illustrate an example of pipeline configuration and instruction codes when the VLIW instruction is executed;

DEPR:
FIG. 1 is a block diagram of a microprocessor 100 embodying the present invention. The microprocessor 100 includes a RISC core section 106 where processing of RISC instructions is performed, a VLIW core section 140 where processing of VLIW instructions is performed and a data cache 129. The technology of manufacturing semiconductor integrated circuits is used to form the microprocessor 100 on one semiconductor substrate of single crystal silicon and the resin molding technique is used for the packaging of the semiconductor substrate by so-called plastic packaging. A RISC instruction is, for example, an instruction to be executed by the RISC type microprocessor described above, whereas a VLIW instruction is, for example, an instruction to be executed by the VLIW type microprocessor described above.

DEPR:
The VLIW core section 140 includes a VLIW start decision unit 104, a VLIW table 113, a VLIW instruction decode unit 115, N pieces of arithmetic units 120 and a VLIW load/store unit 131.

DEPR:
The instruction fetch unit 101 transfers the address of an instruction to be executed to an instruction address bus 105. When a RISC instruction is executed, an instruction corresponding to an address of the instruction address bus 105 is read from the instruction cache 107 to an instruction bus 108 and the RISC instruction is decoded by the RISC instruction decode unit 114. When an applicable instruction is absent (cache miss) in the instruction cache 107, an address bus 134 and a data bus 135 are used to read an instruction from a memory outside the microprocessor via a bus controller (not shown). The decoded result in the RISC decode unit 114 is used to control the floating-point arithmetic unit 118, the integer arithmetic unit 119 and the like through a RISC arithmetic unit control signal 117. Access to the arithmetic units 120 may occur because the arithmetic units 120 of the VLIW core and the integer arithmetic unit 119 as well as the floating-point arithmetic unit 118 of the RISC core are jointly owned as shown in FIG. 1.

DEPR:
When there occurs a branch RISC intion (part of the RISC instruction) for executing the VLIW instruction while the RISC instruction is executed, the instruction fetch unit 101 is notified by the RISC instruction decode unit 114 through a VLIW branch instruction decode signal 142 that the branch instruction has been given. A signal branch destination address is transmitted via a branch destination address bus 136. In a case where the branch destination address is stored in the register of a register file, it is transferred from the integer arithmetic unit 119 via a branch destination address bus 141 to the RISC instruction decode unit 114. The VLIW table 113 is allocated to a specific address space of the memory space of the RISC core and the VLIW instruction stored in the VLIW table 113 is made accessible by gaining access to the address space, whereby the VLIW instruction from the VLIW table 113 can be read and rewritten.

DEPR:
On receiving the notification of the branch instruction through the VLIW branch instruction decode signal 142, the instruction fetch unit 101 notifies the fact that a branch to a VLIW instruction has occurred to the VLIW start decision unit 104 through the VLIW instruction branch signal 103. The VLIW start decision unit 104 decides the condition of activating the VLIW instruction and makes the RISC instruction decode unit 114 suspend the execution of the instruction that is read with the address on the instruction address bus 105 through a RISC core control signal 109. The clock of the whole or part of the RISC core 106 may be stopped so as to reduce power consumption when designated by the register in the RISC instruction decode unit 114 or the VLIW instruction decode unit 115. Even when the VLIW instruction is not executed, the clock of the whole or part of the VLIW core may also be stopped likewise.

DEPR:
The address of the instruction address bus 105 is used to read the VLIW instruction from the VLIW table 113 to a VLIW instruction bus 112. The VLIW instruction decode unit 115 decodes the VLIW instruction on the VLIW instruction bus 112 on the basis of the activating condition transferred from the VLIW start decision unit 104 through a VLIW instruction decode start signal 111, and individually controls the arithmetic units 120 or the integer arithmetic unit 119 and floating-point arithmetic unit 118 of the RISC core section 106 in parallel through a VLIW arithmetic unit control signal 116. The arithmetic units 120 include an integer arithmetic unit, a floating-point arithmetic unit, a DSP (Digital Signal Processor: e.g., product sum computation) dealing with fixed-point data and the like. FIG. 1 shows an example of a local register file type processor when arithmetic units each have their own register files. A transfer of data between arithmetic units is conducted through a computing unit-to-computing unit data bus 123. When the memory access instruction is contained in the VLIW instruction, the VLIW load/store unit 131 has access to the data cache 129 via an address bus 125, a data bus 130 and a data bus 124 for VLIW.

DEPR:
When VLIW instructions are continuously given, the address information held in the VLIW start decision unit 104 via the instruction address bus 105 is used for the VLIW instruction to be read from the VLIW table 113 through an address on an instruction address bus 132 and a VLIW table control signal 133. When the VLIW instruction thus read contains a completion code indicative of the final instruction, the VLIW start decision unit 104 is notified by the VLIW instruction decode unit 115 through a VLIW instruction running state signal 110 that a string of VLIW instructions has been completed. The VLIW instruction running state signal 110 also outputs information which affects the execution of the VLIW instruction such as interruption, exception and the like in addition to the completion of the VLIW instruction. On admitting the completion of the VLIW instruction string, the VLIW start decision unit 104 communicates the completion of the VLIW instruction through a VLIW instruction completion signal 102 and the RISC core control signal 109 to the instruction fetch unit 101 and the RISC instruction decode unit 114, and returns to the RISC instruction.

DEPR:
More specifically, the RISC core section 106 is used for the execution of a RISC instruction, whereas the VLIW core section 140 is used for the execution of a VLIW instruction. The execution of instructions in the VLIW core section 140 is carried out by making the VLIW start decision unit 104 successively read VLIW instructions from the VLIW table 113 that the VLIW start decision unit 104 holds within the microprocessor, with the branch instruction to the VLIW instruction newly added to the RISC instruction or specific address branching as a trigger. After the VLIW instruction is executed, the processing is restored to the RISC instruction string. In other words, VLIW instructions to be executed are dealt with like a subroutine, whereby the processing which needs high-speed action can be executed under the VLIW instruction by storing the instructions in the VLIW table 113. Use of instructions thus reduced to a subroutine makes it possible to maintain compatibility by arranging a string of RISC instructions acting like VLIW instructions at a branch destination even in the case of a microprocessor having no VLIW circuitry.

DEPR:
In a case where action of low power consumption is needed and no high-speed action using the VLIW instruction is needed, the operation of only the RISC core section 106 may be relied upon while the operation of the VLIW core section 140 is suspended, depending on the state of the built-in control register. Conversely in a case where the operation of the RISC core section 106 is not needed, the operation of only the VLIW core section 140 may be relied upon while the operation of the RISC core section 106 is suspended.

DEPR:
It is possible to use resources in common because the computing units (including register files) of the VLIW core section 140 are usable while the RISC instruction is executed and because the computing units (including register files) of the RISC core section 140 are usable while the VLIW instruction is executed. Therefore, the chip area of a semiconductor integrated circuit becomes

reducible.

DEPR:
The VLIW start decision unit 104 decides, on the basis of the decoded result obtained through a control signal 137, whether the VLIW instruction which is being executed is dependent on the RISC instruction as a destination to which the VLIW instruction returns after the VLIW instruction has completely been executed. In the absence of such dependency, selection of control of parallel execution of the RISC and VLIW instructions is possible through the VLIW instruction completion signal 102 and the RISC core control signal 109.

DEPR:
For the VLIW table 113, use can be made of a non-volatile memory such as a masked ROM to be programmed during the process of manufacturing semiconductors, an EPROM (electrically writable ultraviolet ray erasable non-volatile memory), an EEPROM (electrically erasable/writable non-volatile memory), a flash memory (flash EEPROM) and a ferroelectric memory; and a volatile memory such as a DRAM (Dynamic RAM) and an SPRAM (Static RAM). When a rewritable memory, for example, an EEPROM, a flash memory, a ferroelectric memory, a DRAM and a SRAM is employed, the VLIW table 113 corresponding to the system may be set via a VLIW table data bus 122, whereby the VLIW table 113 is made dynamically most suitable for the system during the operation of the microprocessor 100. The VLIW table data bus 122 is also used to provide access to the register in the VLIW start decision unit 104. If SRAM is employed for the VLIW table 113, writing and reading can be carried out at higher speed as compared with the use of any other memory. Since the memory element of DRAM is composed of one transistor while the memory element of SRAM is composed of four to six transistors, a VLIW table using DRAM can be made smaller than when using SRAM. Although the ferroelectric memory is a non-volatile memory, write time can be made substantially equal to read time. Therefore, the use of such a ferroelectric memory makes the rewriting of the VLIW table 113 as quick as that of DRAM.

DEPR:
FIG. 2 shows a modified example of the microprocessor of FIG. 1. What is shown in FIG. 2 is a global register type microprocessor having the registers of the RISC and VLIW cores 106, 140 in common, wherein like reference characters designate like component parts of the microprocessor of FIG. 1. The technology of manufacturing semiconductor integrated circuits is used to form a microprocessor 200 on one semiconductor substrate of single crystal silicon and the resin molding technique is used for the packaging of the semiconductor substrate by so-called plastic packaging.

DEPR:
In the microprocessor 200, a register file 201 is used for read/write operations from and to the floating-point arithmetic unit 118, the integer arithmetic unit 119 and the arithmetic units 120 via a bus 202 between the arithmetic units and the register file. In the case of a memory access instruction, access to a load/store unit 204 occurs via the data bus 121 for RISC and the data bus 124 for VLIW. Data is transferred between the register file 201 and the load/store unit 204 via a bus 203. Data is also transferred between the load/store unit 204 and the data cache 129 via an address bus 206 and a data bus 205. Unlike FIG. 1, FIG. 2 refers to a case where arithmetic units are allowed to control the RISC arithmetic unit control signal 117 and the VLIW arithmetic unit control signal 116 are limited to the arithmetic units in the respective cores.

DEPR:
Since the RISC and VLIW cores 106, 140 have the register file 201, the load/store unit 204 and the data cache 129 in common, the chip area of the semiconductor integrated circuit is reducible.

DEPR:
FIG. 3 is a block diagram illustrating in detail the instruction fetch unit 101, the VLIW table 113, the VLIW start decision circuit 104 and the VLIW instruction decode unit 115 shown in FIGS. 1 and 2.

DEPR:
In the instruction fetch unit 101, the branch condition decision circuit 315 receives the VLIW branch-to-VLIW-instruction decode signal 142 and communicates the occurrence of a branch to the VLIW instruction to the VLIW start decision

circuit 104 through the VLIW instruction branch signal 103. The branch condition decision circuit 315 also decides whether the VLIW instruction has completed through the VLIW instruction completion signal 102 from the VLIW start decision circuit 104. It is thus possible to prevent the VLIW core 140 from being doubly activated during the parallel operation of the VLIW core 140. When such a branch to the VLIW instruction occurs, an address set in the program counter (PC) 316 for the RISC core is transferred form the branch destination address bus 136 to the instruction address bus 105. The program counter 316 for the RISC core is updated by the adder 317.

DEPR:
The VLIW table 113 includes a selector 303, an address latch 304, a data latch 305 and a VLIW table storage memory 306. In the VLIW table 113, the selector 303 is used to select an address set by the instruction fetch unit 101 via the instruction address bus 105 and an address set by a program counter 302 for VLIW held in the VLIW start decision circuit 104 via the instruction address bus 132. The address thus selected is set at the address latch 304 and used to transfer the VLIW instruction from the VLIW table storage memory 306 to the VLIW instruction bus 112. The VLIW table is changed by means of the data transferred to the VLIW table data bus 122 via the data latch 305 with respect to the VLIW table storage memory 306 indicative of the address designated by the address latch 304.

DEPR:
The VLIW start decision circuit 104 includes the VLIW program counter 302 of holding the address on the instruction address bus 105, an adder 301 for incrementing the program counter 302, a decoder 309 for decoding the address from the program counter 302, a VLIW core control register (REG) for holding information as to acceptability of VLIW activation and what is necessary for VLIW activation, a VLIW start decision circuit 310 for deciding the activation of VLIW on receiving information about the signal 103 informing a branch to the VLIW instruction based on the decoded result, a value of a VLIW core control register 308 and the VLIW instruction running state signal 110, and the VLIW table storage memory 307 for controlling the VLIW table according to information from the VLIW core control register 308. In this case, the VLIW core control register 308 is updated via the VLIW table data bus 122.

DEPR:
The VLIW instruction decode unit 115 includes a multiplexer 313 for selecting the VLIW instruction from the VLIW instruction bus 112, an adjusting circuit 311 for deciding execution control on receiving information about a VLIW instruction completion code on the VLIW instruction bus 112 and an RISC instruction code on a control signal 137, a starting circuit 312 for controlling the activation of VLIW computing units on receiving the VLIW instruction decode start signal 111, and a VLIW arithmetic unit control circuit 314 for controlling a plurality of arithmetic units prepared for VLIW.

DEPR:
As shown in FIG. 4, in contrast to FIG. 3, an address translation buffer 401 is situated between the instruction fetch unit 101 and the VLIW table 113, wherein like reference characters designate like component parts of FIG. 3.

DEPR:
The address translation buffer 401 is used to translate an instruction address (logical address) 105 into an instruction address (physical address) 403, which is transferred to the instruction cache 107, the VLIW table 113 and the VLIW start decision circuit 104. When the instruction address is translated in the address translation buffer 401, a VLIW start auxiliary information signal 402 is read out and utilized for the operation of the VLIW start decision circuit 310 and the stating circuit 312.

DEPR:
FIG. 5A shows a timing chart and an example of VLIW instruction code allocation in a case where the VLIW and RISC instructions are executed in parallel by the VLIW branch instruction.

DEPR:
The VLIW branch instruction is a sort of RISC instruction and like RISC instructions 1, 2, 3, executed through the pipeline processing in the RISC core

section 106 and reference numerals 501, 504, 506 and 507 denote their pipeline stages. Namely, I represents an instruction fetch; D, an instruction code; E, execution; M, memory access; and W, a write stage. Further, VLIW instructions 1, 2, 3 are executed through the pipeline processing and reference numerals 502, 503, 505 denote their pipeline stages; namely, VI, VD, VE, VM, VW represent an instruction fetch for VLIW, an instruction decode, execution, memory access and a write stage, respectively.

DEPR:
Assuming that a branch to the VLIW instruction 1 occurs three cycles after the VLIW branch instruction, the pipeline 502 of the VLIW instruction 1 starts at a cycle 4. When the VLIW instruction which follows the VLIW instruction 2 is not dependent on the RISC instruction at the return destination after the execution of the VLIW instruction, it is possible to carry out the execution 503 of the VLIW instruction 2 and the execution 504 of the RISC instruction 1 in parallel. The execution 505 of the VLIW instruction 3 and the execution 506 of the RISC instruction 2 may also be carried out in parallel and when the VLIW instruction is completed, the pipeline for the VLIW instruction disappears and only the execution 507 of the RISC instruction 3 is carried out.

DEPR:
An example of the code allocation of the VLIW instruction read from the VLIW table at the VI stage is shown in FIG. 5B. A code for controlling the plurality of computing units is buried in one VLIW instruction. The instruction length of the VLIW instruction is a fixed length of 128 or 256 bits. Reference numeral 508 denotes a field for controlling the VLIW computing units 1; 509, a field for controlling the VLIW computing units 2; 510, a field for controlling the integer computing units; 511, a field for controlling the floating-point computing units; and 512, a field for storing the completion of the VLIW instruction and other items of information. Each of the fields 508, 509, 510, 511, 512 is 32-bit long and the operational code of 513, the displacement of 514, the source register 1 of 515, the source register 2 of 516 and the destination register of 517 are allocated. A VLIW decoder decodes this instruction code so as to control each computing units.

DEPR:
FIG. 6 is a general view of a system utilizing the present invention. Reference numeral 601 denotes a system built on one or a plurality of printed boards to which a display 611, a CD-ROM (Compact Disc-ROM) 612, a video camera 615, an LCD (Liquid Crystal Display) 618 and peripheral devices 620 such as a keyboard and a printer are connected. The system 601 includes a processor 602, a ROM 608 for storing programs, an SRAM 609 for storing data and programs, a DRAM 614 and an SDRAM (Synchronous DRAM) 614, a three-dimensional graphic frame buffer 610 for storing display image data and the like, a CD-ROM decoder 613 for controlling the CD-ROM, a video controller 616 for controlling the video camera, a LCD controller 619 for controlling a liquid crystal display (LCD), and a peripheral device controller 621 for controlling the peripheral devices 620 such as the keyboard and the printer, these component parts being connected to a system bus 607.

DEPR:
When the processor 1 (708) and the processor 2 (711) are used in combination, a combination of a RISC processor with a 16-bit fixed length instruction and a VLIW processor with a 64-bit fixed length instruction, a combination of a RISC processor with a 32-bit fixed length instruction and a VLIW processor with a 128-bit fixed length instruction, a combination of a RISC processor with a 64-bit fixed length instruction and a VLIW processor with a 256-bit fixed length instruction or the like may be used. In these combinations, the processors 1 and 2 may be reversed in order. FIG. 7 refers to a case where the processor 1 (708) is a RISC processor (RISC core 106 of FIG. 1), and the processor 2 (711) a VLIW processor (VLIW core 140 of FIG. 1).

DEPR:
The processor 1 (708) includes the instruction fetch unit 101, the instruction cache 107, the RISC instruction decode unit 114, one or a plurality of arithmetic units 704 and the RISC load/store unit 127, these being mutually connected by an internal bus 702. The instruction fetch unit 101 reads an instruction from the instruction cache 107 and the RISC instruction decode unit 114 analyzes the instruction thus read, causing the arithmetic units 704 to operate, whereby the RISC load/store unit 127 is activated whenever memory access is needed. A

multiplication unit, an integer arithmetic unit, a floating-point unit, a substraction unit, a DSP and so forth are placed in the arithmetic unit 704. The arithmetic unit may have a plurality of similar arithmetic units. A bus 709 is used to input an instruction to the instruction cache from a bus control unit 712. The RISC load/store unit 127 uses the bus 705 to access the data cache 129 and the bus control unit 712 uses the bus 707 to fetch data into the data cache 129.

DEPR:
The processor 2 (711) includes the VLIW start decision circuit 104, the VLIW table 113, the VLIW instruction decode unit 115, one or a plurality of arithmetic units 704 and the VLIW load/store unit 131, these being mutually connected by the internal bus 702. When the activation of the VLIW instruction is decided by the VLIW start decision circuit 104, it is read from the VLIW table 113 and analyzed in the VLIW instruction decode unit 115 and the arithmetic units 704 operate, whereby the VLIW load/store unit 131 is activates whenever memory access is needed. For the VLIW table 113, use can be made of a non-volatile memory such as a masked ROM and a flash memory or a volatile memory such as a SRAM and a DRAM. When the VLIW table is located outside the chip, the VLIW instruction is read from outside via a bus control unit. A multiplication unit, an integer arithmetic unit, a floating-point arithmetic unit, a substraction unit, a DSP and so forth are placed in the arithmetic unit 704. The arithmetic unit may have a plurality of similar arithmetic units. The VLIW load/store unit 131 uses a bus 706 to gain access to the data cache 129.

DEPR:
The first processor 801 is a RISC type processor for executing an instruction of 16-bit fixed length. The second processor 802 is a VLIW type processor for executing an instruction of 64-bit length. The first and second processors 801, 802 are different in instruction system (instruction set, instruction format, programming model or the like). An instruction (program) to be executed by the first processor 801 is stored in the first memory 803, whereas an instruction (program) to be executed by the second processor 802 is stored in the second memory 804. The first and second memories 803, 804 are non-volatile or volatile memories. By making the first and second memories 803, 804 rewritable memories, not only the semiconductor supplier but also semiconductor users are able to write programs. In other words, users are allowed to construct both programs using a first and a second instruction set. Volatile, non-volatile and rewritable memories are similar to those used in the VLIW table 113 of the microprocessor of FIG. 1.

DEPR:
Since the instruction length in the first and second processors 801, 802 is relatively short, the control unit such as an instruction decoder and the arithmetic unit can be small-sized, so that the capacity of the first memory 803 or the second memory 804 is increasable. In a case where the area occupied by the processor is not critical, the instruction length may be increased.

DEPR:
The foregoing invention has been described in detail in terms of preferred embodiments. However, the present invention is not limited to those embodiments but may be practiced in various modified ways without departing from the spirit and scope thereof. Although the microprocessor of FIG. 1, for example, is configured so that the computing units of the RISC and VLIW cores are arranged for common use mutually, they may be used independently as in the microprocessor of FIG. 2. FIG. 2 refers to a case where the load/store unit 204 is commonly used by the RISC and VLIW cores; however, there may be provided two load/store units for both the cores, respectively. Two processors having different instruction sets may be arranged instead of providing the RISC type processor as the first processor and the VLIW type processor as the second processor. In other words, processors of RISC and RISC types, RISC and CISC types, CISC and CISC types, CISC and VLIW types, VLIW and VLIW types or the like. Although combinations of 16 bits and 64 bits, 32 bits and 128 bits, and 64 bits and 256 bits have been shown to constitute the instruction lengths of the RISC processor and the VLIW processor by way of example, instruction lengths are not limited to the combinations above. Moreover, new embodiments may be formed according to the present invention by replacing the whole or part of the combinations of the instruction lengths if necessary.

DEPL:
Detailed Description of VLIW Core

DEPL:
Modified Example of VLIW Core

CLPR:
8. A data processor according to claim 7, wherein the VLIW instructions include an instruction for three-dimensional graphic processing.

CLPR:
10. A microprocessor according to claim 7, wherein a non-volatile memory has a built-in VLIW table.

CLPR:
12. A microprocessor according to claim 7, wherein a volatile memory has a built-in VLIW table.

CLPR:
13. A microprocessor according to claim 12, wherein the volatile memory of said VLIW table is one of a SRAM and a DRAM.

CLPR:
14. A microprocessor according to claim 12, wherein a memory rewritable with the processor of the RISC core has a built-in VLIW table.

CLPR:
15. A data processor according to claim 7, wherein the RISC instructions have a variable length and the VLIW instructions have a fixed length.

CLPR:
16. A data processor according to claim 7, wherein the RISC instructions and the VLIW instructions each have a fixed length.

CLPR:
17. A data processor according to claim 16, wherein a length of the RISC instructions is shorter than a length of the VLIW instructions.

CLPR:
20. A microprocessor according to claim 18, wherein a non-volatile memory has a built-in VLIW table.

CLPR:
22. A microprocessor according to claim 18, wherein a volatile memory has a built-in VLIW table.

CLPR:
23. A microprocessor according to claim 22, wherein the volatile memory of said VLIW table is one of a SRAM and a DRAM.

CLPR:
24. A microprocessor according to claim 22, wherein a memory rewritable with the processor of the RISC core has a built-in VLIW table.

CLPR:
25. A data processor according to claim 18, wherein the RISC instructions have a variable length and the VLIW instructions have a fixed length.

CLPR:
26. A data processor according to claim 18, wherein the RISC instructions and the VLIW instructions each have a fixed length.

CLPR:
27. A data processor according to claim 26, wherein a length of the RISC instructions is shorter than a length of the VLIW instructions.

CLPR:
28. A data processor according to claim 18, wherein the VLIW instructions include an instruction for three-dimensional graphic processing.

CLPR:
31. A microprocessor according to claim 29, wherein a non-volatile memory has a built-in VLIW table.

CLPR:
33. A microprocessor according to claim 29, wherein a volatile memory has a built-in VLIW table.

CLPR:
34. A microprocessor according to claim 33, wherein the volatile memory of said VLIW table is one of a SRAM and a DRAM.

CLPR:
35. A microprocessor according to claim 33, wherein a memory rewritable with the processor of the RISC core has a built-in VLIW table.

CLPR:
36. A data processor according to claim 29, wherein the RISC instructions have a variable length and the VLIW instructions have a fixed length.

CLPR:
37. A data processor according to claim 29, wherein the RISC instructions and the VLIW instructions each have a fixed length.

CLPR:
38. A data processor according to claim 37, wherein a length of the RISC instructions is shorter than a length of the VLIW instructions.

CLPR:
39. A data processor according to claim 29, wherein the VLIW instructions include an instruction for three-dimensional graphic processing.

CLPR:
42. A microprocessor according to claim 40, wherein a non-volatile memory has a built-in VLIW table.

CLPR:
44. A microprocessor according to claim 40, wherein a volatile memory has a built-in VLIW table.

CLPR:
45. A microprocessor according to claim 44, wherein the volatile memory of said VLIW table is one of a SRAM and a DRAM.

CLPR:
46. A microprocessor according to claim 44, wherein a memory rewritable with the processor of the RISC core has a built-in VLIW table.

CLPR:
47. A data processor according to claim 40, wherein the RISC instructions have a variable length and the VLIW instructions have a fixed length.

CLPR:
48. A data processor according to claim 40, wherein the RISC instructions and the VLIW instructions each have a fixed length.

CLPR:
49. A data processor according to claim 48, wherein a length of the RISC instructions is shorter than a length of the VLIW instructions.

CLPR:
50. A data processor according to claim 40, wherein the VLIW instructions include an instruction for three-dimensional graphic processing.

CLPR:
54. A data processor according to claim 53, wherein the RISC instructions have a variable length and the VLIW instructions have a fixed length.

CLPR:
55. A data processor according to claim 53, wherein the RISC instructions and the VLIW instructions each have a fixed length.

CLPR:
56. A data processor according to claim 55, wherein a length of the RISC instructions is shorter than a length of the VLIW instructions.

CLPR:
57. A data processor according to claim 53, wherein the VLIW instructions include an instruction for three-dimensional graphic processing.

CLPR:
59. A data processor according to claim 58, wherein the RISC instructions have a variable length and the VLIW instructions have a fixed length.

CLPR:
60. A data processor according to claim 58, wherein the RISC instructions and the VLIW instructions each have a fixed length.

CLPR:
61. A data processor according to claim 60, wherein a length of the RISC instructions is shorter than a length of the VLIW instructions.

CLPR:
62. A data processor according to claim 58, wherein the VLIW instructions include an instruction for three-dimensional graphic processing.

CLPV:
a reduced instruction set computer (RISC) core for executing a RISC instruction, a very long instruction word (VLIW) table for storing VLIW instructions, a VLIW core for executing a VLIW instruction, and a VLIW start decision unit for controlling operation switching between said RISC core and said VLIW core,

CLPV:
wherein when said RISC core executes a subroutine call or a branch instruction out of RISC instructions, a VLIW instruction is read from said VLIW table by use of a branch destination address, operation is transferred by said VLIW start decision unit from said RISC core to said VLIW core, and said VLIW core is caused to operate until a VLIW instruction completion code is read from said VLIW table; and

CLPV:
a circuit for examining whether an operation of the VLIW instruction executed subsequent to a branch to the VLIW instruction is relevant to an operation of the RISC instruction which is restored after the execution of the VLIW instruction, said microprocessor being capable of executing the RISC instruction at the restoring destination in parallel without waiting for the completion of the VLIW instruction when no relevance exists.

CLPV:
an address translation table for subjecting the branch destination address to address translation so that a signal for expanding the VLIW instruction read from said VLIW table may be read from the address translation table simultaneously when the address translation is conducted.

CLPV:
a reduced instruction set computer (RISC) core for executing a RISC instruction, a very long instruction word (VLIW) table for storing VLIW instructions, a VLIW core for executing a VLIW instruction, and a VLIW start decision unit for controlling operation switching between said RISC core and said VLIW core,

CLPV:
a register for storing information which is used to prohibit and control the execution of the VLIW instruction in a processor; and

CLPV:
a function of lowering power consumption by causing a branch to a string of RISC instructions performing an operation equivalent to the VLIW instruction at the

time execution of the VLIW instruction is prohibited so as to suspend the operation of the VLIW core.

CLPV:
an address translation table for subjecting the branch destination address to address translation so that a signal for expanding the VLIW instruction read from said VLIW table may be read from the address translation table simultaneously when the address translation is conducted.

CLPV:
a reduced instruction set computer (RISC) core for executing a RISC instruction, a very long instruction word (VLIW) table for storing VLIW instructions, a VLIW core for executing a VLIW instruction, and a VLIW start decision unit for controlling operation switching between said RISC core and said VLIW core,

CLPV:
a register for storing information which is used to control the execution of VLIW instructions in the processor,

CLPV:
an address translation table for subjecting the branch destination address to address translation so that a signal for expanding the VLIW instruction read from said VLIW table may be read from the address translation table simultaneously when the address translation is conducted.

CLPV:
a reduced instruction set computer (RISC) core for executing a RISC instruction, a very long instruction word (VLIW) table for storing VLIW instructions, a VLIW core for executing a VLIW instruction, and a VLIW start decision unit for controlling operation switching between said RISC core and said VLIW core,

CLPV:
a register for storing information which is used to control the execution of VLIW instructions in the processor,

CLPV:
an address translation table for subjecting the branch destination address to address translation so that a signal for expanding the VLIW instruction read from said VLIW table may be read from the address translation table simultaneously when the address translation is conducted.

CLPW:
wherein when said RISC core executes a subroutine call or a branch instruction out of RISC instructions, a VLIW instruction is read from said VLIW table by use of a branch destination address, operation is transferred by said VLIW start decision unit from said RISC core to said VLIW core, and said VLIW core is caused to operate until a VLIW instruction completion code is read from said VLIW table;

CLPW:
wherein when said RISC core executes a subroutine call or a branch instruction out of RISC instructions, a VLIW instruction is read from said VLIW table by use of a branch destination address, operation is transferred by said VLIW start decision unit from said RISC core to said VLIW core, and said VLIW core is caused to operate until a VLIW instruction completion code is read from said VLIW table; and

CLPW:
wherein the execution of the VLIW instruction is prohibited if said register stores information for prohibiting the execution of the VLIW instruction.

CLPW:
wherein when said RISC core executes a subroutine call or a branch instruction out of RISC instructions, a VLIW instruction is read from said VLIW table by use of a branch destination address, operation is transferred by said VLIW start decision unit from said RISC core to said VLIW core, and said VLIW core is caused to operate until a VLIW instruction completion code is read from said VLIW table; and

CLPW:
wherein the execution of the VLIW instruction is prohibited and a plurality of RISC instructions equivalent to the VLIW instructions are executed if said register stores information for prohibiting the execution of the VLIW instructions.

CLPW:
wherein the execution of the instruction of the second group of instructions in the processor is prohibited and a plurality of the instructions of the first set of instructions equivalent to the instruction of the second group of instructions are executed if said register stores information prohibiting the execution of the instruction of the second group of instructions in the processor.

CLPW:
wherein the execution of the instruction of the second group of instructions in the processor is prohibited and a plurality of the instructions of the first set of instructions equivalent to the instruction of the second group of instructions are executed if said register stores information prohibiting the execution of the instruction of the second group of instructions in the processor.

Generate Collection

Term	Documents
DECODER$	0
DECODER.USPT.	72011
DECODERAND.USPT.	1
DECODERARRAY.USPT.	1
DECODERBAR.USPT.	1
DECODERBLOCK.USPT.	1
DECODERCOMPLIANCE.USPT.	1
DECODERCONTROLLER.USPT.	7
DECODERCORE.USPT.	1
DECODERDCLR.USPT.	1
(L13 AND DECODER$ AND VLIW).USPT.	1

There are more results than shown above. Click here to view the entire set.

Display 10 Documents, starting with Document: 1

Display Format: KWIC Change Format

L13: Entry 12 of 15 File: USPT Jan 12, 1993

DOCUMENT-IDENTIFIER: US 5179702 A
TITLE: System and method for controlling a highly parallel multiprocessor using an anarchy based scheduler for parallel execution thread scheduling

BSPR:
The executable form of a multithreaded program consists of multiple threads that can be executed in parallel. In the operating system, the representation of the executable form of a program is a process. A process executes a single thread of a program during a single time period. Multiple processes can each execute a different thread or the same thread of a multithreaded program. When multiple processes executing multiple threads of a multithreaded program are simultaneously executing on multiple processors, then parallel processing of a program is being performed. When multiple processes execute multiple threads of a multithreaded program, the processes share a single process image and are referred to as shared image processes. A process image is the representation in the operating system of the resources associated with process. The process image includes the instructions and data for the process, along with the execution context information for the processor (the values in all of the registers, both control registers and data registers, e.g., scalar registers, vector registers, and local registers) and the execution context information for operating system routines called by the process.

BSPR:
In an effort to increase the processing speed and flexibilty of supercomputers, the cluster architecture for highly parallel multiprocessors described in the previously identified parent application provides an architecture for supercomputers wherein multiple processors and external interfaces can make multiple and simultaneous requests to a common set of shared hardware resources, such as main memory, global registers and interrupt mechanisms. Although this new cluster architecture offers a number of solutions that can increase the parallelism of supercomputers, these solutions will not be utilized by the vast majority of users of such systems without software that implements parallelism by default in the user environment and provides an operating system that is fully capable of supporting such a user environment. Accordingly, it is desirable to have a software architecture for a highly parallel multiprocessor system that can take advantage of the parallelism in such a system.

BSPR:
The primary mechanism for integrating the multithreaded operating system and the parallel user environment is a set of data structures referred to as the Operating System Shared Resources (OSSR) which are defined in relation to the various hardware shared resources, particularly the common shared main memory and the global registers. The OSSRs are used primarily by the operating system, with a limited subset of the OSSRs available to the user environment. Unlike prior art operating systems for multiprocessors, the OSSRs are accessible by both the processors and external interface ports to allow for a distributed input/output architecture in the preferred multiprocessor system. A number of resource allocation primitives are supported by the hardware shared resources of the preferred embodiment and are utilized in managing the OSSRs, including an atomic resource allocation mechanism that operates on the global registers.

DEPR:
Referring now to FIG. 3, a simplified representation of the relative amounts of context switch information is shown for the three types of context switches: lightweight processes, intra-process group switches and inter-process group switches. Based upon this representation, it is easy to understand that the best way to minimize total context switch overhead is to have the majority of context switches involve lightweight processes. Unfortunately, as shown in FIG. 4a, the prior art scheduling of lightweight processes is a cumbersome one-way technique

wherein the user program determines the type of lightweight processes it wants to have scheduled based on its own independent criteria using data structures in the main memory that are unrelated to the other operating system scheduling that may be occurring in the multiprocessor. Because the user-side scheduling of such lightweight processes and the operating system are not integrated, the context switch overhead for lightweight process context switches is increased. In the present invention, shown in FIG. 4b, both the user-side scheduler and the operating system operate on the same set of OSSR's that use both shared common memory and global registers. As a result, there is a two-way communication between the operating system and the user-side scheduler that allows the present invention to decrease the context switch overhead associated with lightweight processes, and in particular, with a new type of lightweight process referred to as a microprocess.

DEPR:
Referring now to FIG. 5, a single multiprocessor cluster of the preferred embodiment of the multiprocessor cluster system for executing the present invention is shown having a plurality of high-speed processors 10 sharing a large set of shared resources 12 (e.g., main memory 14, global registers 16, and interrupt mechanisms 18). In this preferred embodiment, the processors 10 are capable of both vector and scalar parallel processing and are connected to the shared resources 12 through an arbitration node means 20. The processors 10 are also connected through the arbitration node means 20 and a plurality of external interface ports 22 and input/output concentrators (IOC) 24 to a variety of external data sources 26. The external data sources 26 may include a secondary memory system (SMS) 28 linked to the input/output concentrator means 24 via one or more high speed channels 30. The external data source 26 may also include a variety of other peripheral devices and interfaces 32 linked to the input/output concentrator via one or more standard channels 34. The peripheral device and interfaces 32 may include disk storage systems, tape storage systems, terminals and workstations, printers, and communication networks.

DEPR:
The multiprocessor cluster system of the preferred embodiment creates a computer processing environment in which parallelism is favored. Some of mechanisms in the multiprocessor cluster system which aid the present invention in coordinating and synchronizing the parallel resources of such a multiprocessor system include, without limitation: the distributed input/output subsystem, including the signaling mechanism, the fast interrupt mechanism, and the global registers and the atomic operations such as TAS, FAA, FCA and SWAP that operate on the global registers as described in greater detail in the previously identified co-pending application entitled DISTRIBUTED INPUT/OUTPUT ARCHITECTURE FOR A HIGHLY PARALLEL MULTIPROCESSOR SYSTEM Ser. No. 07/536,182; the mark instructions, the load instruction, the accounting registers and watchpoint addresses as described in greater detail in the previously identified parent application entitled CLUSTER ARCHITECTURE FOR A HIGHLY PARALLEL MULTIPROCESSOR SYSTEM Ser. No. 07/459,083; and the various mechanism that support the pipelined operation of the processors 10, including the instruction cache and the separate issue and initiation of vector instructions as described in greater detail in the previously identified co-pending application entitled SCALAR/VECTOR PROCESSOR Ser. No. 07/536,409. Together, and individually, these mechanisms support the symmetric access to shared resources and the multi-level pipeline operation of the preferred multiprocessor system.

DEPR:
The primary mechanism for integrating the multithreaded operating system 1000 and the parallel user environment 2000 is a set of data structures referred to as the Operating System Shared Resources (OSSR) 2500 which are defined in relation to the various hardware shared resources 12, particularly the common shared main memory 14 and the global registers 16. The OSSR 2500 is a set of data structures within the SSI/mOS 1000 that define the allocation of global registers 16 and main memory 14 used by the operating system 1000, the parallel user environment 2000, the distributed input/output architecture via the external interfaces 22 and the main memory 14.

DEPR:
When a shared image process group is created, part of context of the shared image process group is a dynamically allocated set of global registers that the shared image process group will use. Each shared image process group is allocated one or

more work request queues in the set of global registers. In the preferred embodiment, the sets of global registers are defined by the operating system in terms of absolute addresses to the global registers 16. One of the global registers is designated as the total of all of the outstanding help requests for that shared image process group. By convention, the help request total is assigned to GO in all sets of global registers. In the situation where the processor looking for work is executing a microprocess or a process that is assigned to the same shared image process group as the global register with the help request total (i.e., intra-process context switch), the resulting switch overhead is minimal as no system related context expense is required to perform the requested work. If the processor looking for work in a given help request total (GO) is executing a microprocess not assigned to the same shared image process group, the processor executing the microprocess must first acquire the necessary microprocess context of the shared image process group for this global register set before examining the help request queues.

DEPR:
Cooperating Process - This term is used to identify those processes that are sharing (and are thus synchronizing through) a single set of global registers. This means the value in the global register control register is the same for each cooperating process. By default, each microprocess is a cooperating process with its respective initiating process. Shared image processes may or may not be cooperating processes, although by default they are. Through the use of system calls, non-shared image processes can become cooperating processes.

DEPR:
Processor Context - Each process has processor context. In the preferred embodiment, processor context includes the scalar, vector, and global registers being used by the process or microprocess, plus the control register settings that currently dictate the execution environment. To allow the process to continue executing at its next scheduling interval, a subset of this processor context is saved across interrupts, exceptions, and traps. Exactly what is saved depends on the source of the event triggering the context switch.

DEPR:
In the present invention, microprocesses are created as an automatically multithreaded program is executed. An existing process posts a request in a global register asking that a microprocess or microprocesses be made available. At this point, any available processor can be used as a microprocess. It will be noted that System V mechanisms can also create microprocesses, iprocs, kprocs, and shared image processes as well as traditional System V processes using the present invention.

DEPR:
The dispatcher 1112 manages the progress of processes through the states as shown in FIG. 12. Processors 10 use the dispatcher portion of SSI/mOS to check for the highest priority process or microprocess that is ready to run. Kprocs, iprocs, and mprocs will each be a separate scheduling class. Requests by a process (usually a shared image process group) for work to be scheduled will increment a value in one of the global registers 16 that is associated with that process. The specified global register is chosen by convention as described in greater detail hereinafter and will be referred to for the description of FIG. 12 as the Help Request (HRR). The increment of the HRR is an atomic action accomplished by use of one of the atomic resource allocation mechanisms associated with the OSSR's. At state 1163, the operating system 1000 has a processor 10 that can be scheduled to do new work. Based on site selectable options, the operating system can either (1) always choose to schedule processes first to state 1162 for traditional process scheduling and only if no executable process is found check the HRR in state 1165; or (2) always schedule some portion of the processors 10 to check the HRR in state 1165 to support parallel processing (and, in particular, the processing of mprocs) and schedule the remainder of the processors 10 to state 1162 for traditional process scheduling. This assignment balance between state 1162 and 1165 is modified in real time in accordance with a heuristic algorithm to optimize the use of the multiprocessor system based on predictive resource requirements obtained from the accounting registers in the processor 10. For example, all other conditions being equal, an available processor will be assigned to threads executing at the highest computation rate, i.e. the most efficient processors.

DEPR:
Processes that are sent to state 1165 and do not have any context to be saved at the point they reach state 1165 can become microprocesses. In state 1165, the microprocesses examine each group of global registers assigned to a shared image process group and, specifically, examine the HRR global register for that shared image process group. If the HRR register is positive, then the shared image process group has requested help. The microprocess automatically decrements the count in the HRR (thus indicating that one of the request made to the HRR has been satisfied) and proceeds to state 1169 for scheduling by the User-Side Scheduler.

DEPR:
The cluster architecture supports multiple input/output streams, thus supporting disk striping and multiple simultaneous paths of access to the Secondary Memory System (SMS). The input/output concentrator (IOC) 24 distributes work across processors 10 and input/output logical devices 30. Low level primitives in the IOC 24 expose memory 14, SMS 28, and the global registers 16 to the system's device controllers. All the SMS transfer channels can be busy at the same time. The cluster architecture also provides an optional expanded caching facility through the high bandwidth SMS, using the SMS to cache.

DEPR:
As a result of the HiPPI channel implementation, peripheral devices have a pipe directly into main memory that is similar to a DMA component. Devices also have direct access to the multiprocessor system's global registers and operations, and they are able to directly read/write data to the SMS system. As a result, software running on peripheral devices can in most cases be considered as a logical extension of the operating system. Some implications of the input/output channel design are that peripheral device controllers have to be intelligent and programmable, and they must implement a low level HiPPI command protocol.

DEPR:
The distributed device drivers of the present invention are written so that some typical driver functions will execute out in the peripheral device controller. This model is sometimes preferable, and is made possible by the fact that peripheral device controllers have access to kernel signals, main memory, secondary memory, and global registers.

DEPR:
Ranges during which the value of a variable is kept in a register (as opposed to the memory location of the variable) are maintained by the compiler for use by the debugger. This provides the debugger with the location of the current value of a variable. In each basic block in a procedure a variable is assigned to at most one register. For each basic block the compiler keeps a list (logically) of variables and associated registers. This information is generated as part of the local register allocation phase of the compile and is kept for the debugger.

CLPV:
~~control means~~ for distributively controlling the operation and execution of a plurality of multithreaded programs in the multiprocessor system by executing a symmetrically integrated multithreaded operating system program on one or more of the processors that has an anarchy-based scheduling model for scheduling one or more processes and resources associated with each multithreaded program for execution on one or more of the processors, each processor having access to a single image of the operating system program stored in the common memory that operates on a common set of operating system shared resources, the operating system programming means comprising:

CLPW:
input/output means for processing distributed, multithreaded input/output services such that any one or more of the input/output services can be executed concurrently by multiple ones of the processors and a plurality of input/output resources associated with the multiprocessor system, the input/output resources including a plurality of peripheral devices attached to the multiprocessor system via a plurality of external interfaces, the input/output services including

CLPW:
(a) processing a plurality of multithreaded system services such that any one or more of the system services can be executed concurrently by multiple processors,

the system services including:

CLPW:
(b) processing a plurality of multithreaded input/output services such that any one or more of the input/output services can be executed concurrently by multiple ones of the processors and a plurality of input/output resources associated with the multiprocessor system, the input/output resources including a plurality of peripheral devices attached to the multiprocessor system via a plurality of external interfaces, the input/output services including